Informe: Detección y Mitigación de Múltiples Vulnerabilidades Web

1. Objetivo del Ejercicio

El objetivo de este ejercicio fue analizar un fragmento de código PHP de un sistema de comentarios para identificar, mitigar y validar amenazas de tipo SQL Injection, Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF) utilizando buenas prácticas y herramientas de seguridad como OWASP ZAP.

2. Análisis de Vulnerabilidades

Se analizó el siguiente fragmento de código PHP:

```
PHP
<?php
$nombre = $_POST['nombre'];
$comentario = $_POST['comentario'];
$query = "INSERT INTO comentarios (nombre, comentario) VALUES ('$nombre',
'$comentario')";
mysqli_query($conexion, $query);
?>
```

a. Vulnerabilidades Identificadas: Las vulnerabilidades identificadas en este código son:

- **Inyección SQL:** Los valores de $nombre y $comentario se concatenan directamente en la consulta SQL sin sanitización ni uso de prepared statements, permitiendo a un atacante inyectar código SQL malicioso.
- **XSS (Cross-Site Scripting):** Los datos de $nombre y $comentario se insertan en la base de datos sin escape y, si se muestran directamente en una página web, permitirían la ejecución de scripts maliciosos en el navegador del usuario.
- **CSRF (Cross-Site Request Forgery):** La falta de un token anti-CSRF en el formulario permite que un atacante fuerce a un usuario autenticado a realizar acciones no deseadas.

b. Ataque XSS en el campo comentario: Un ataque XSS que se podría ejecutar en el campo comentario es: <script>alert('XSS');</script>. Si este string se almacena y luego se renderiza sin escape, el script se ejecutaría en el navegador del usuario.

c. Prevención de XSS: Para evitar el XSS en este sistema, la técnica correcta es **Escapar caracteres especiales con htmlspecialchars()**. Esta función debe aplicarse al momento de la salida de los datos hacia el HTML.

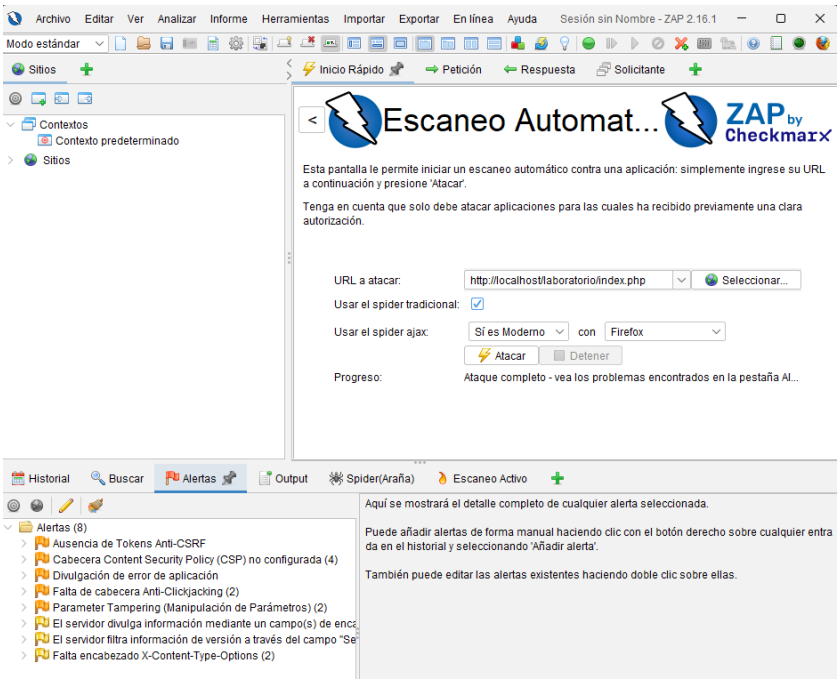
3. Corrección del Código

Se reescribió el código PHP y el formulario HTML para mitigar las vulnerabilidades identificadas, aplicando las siguientes técnicas:

- **Evitar Inyección SQL:** Se implementaron **Prepared Statements** con mysqli_prepare() y mysqli_bind_param() para separar la lógica de la consulta de los datos de entrada, garantizando que los datos sean tratados como tales y no como código SQL.
- **Prevenir XSS:** Se utilizó la función htmlspecialchars() de PHP para escapar caracteres especiales en las entradas ($nombre y $comentario) antes de su inserción y, de manera crucial, antes de mostrarlos en la interfaz web, convirtiendo los caracteres HTML en entidades y previniendo la ejecución de scripts maliciosos.
- **Incluir un token CSRF:** Se implementó un **token CSRF** único por sesión. Este token se genera en el servidor, se almacena en la sesión del usuario ($_SESSION['csrf_token']), se incluye como un campo oculto en el formulario HTML y se valida en el servidor antes de procesar la solicitud POST. Si los tokens no coinciden, la solicitud es rechazada.

4. Evaluación Práctica con Herramienta

Se utilizó **OWASP ZAP** en un entorno controlado para evaluar tanto el código vulnerable original como la versión corregida, validando la presencia y posterior mitigación de las vulnerabilidades.



1. **Hallazgo: Inyección SQL Detectada**
    - **Descripción del Hallazgo con ZAP:** OWASP ZAP detectó una vulnerabilidad de "SQL Injection" al escanear el formulario de comentarios. Esto se evidencia por la alerta generada por la herramienta, que indica que los parámetros de entrada podrían ser manipulados para ejecutar comandos SQL maliciosos.
    - **Corrección Aplicada:** Esta vulnerabilidad se mitigó mediante la implementación de *Prepared Statements* en el código PHP, asegurando que los datos sean tratados como valores y no como parte de la consulta SQL.

```php
// Usar prepared statements para prevenir SQLi
$stmt = $conexion->prepare("INSERT INTO comentarios (nombre, comentario) VALUES (?, ?)")
$stmt->bind_param("ss", $nombre, $comentario);
$stmt->execute();
```

2. **Hallazgo: XSS (Cross-Site Scripting Persistente) Detectado**
    - **Descripción del Hallazgo con ZAP:** OWASP ZAP también reportó una alerta de "Cross Site Scripting (Persistent)". Esto confirmó que la aplicación era susceptible a la inyección de scripts a través de los campos de entrada, los cuales, al no ser correctamente sanitizados o escapados al mostrarse, podrían ejecutarse en el navegador de los usuarios finales.
    - **Corrección Aplicada:** La prevención de XSS se logró escapando los caracteres especiales HTML (<, >, etc.) utilizando la función htmlspecialchars() de PHP, tanto antes de la inserción en la base de datos como, de forma crucial, al momento de la salida de los datos en la interfaz web.

```php
// Sanitizar entradas para prevenir XSS
$nombre = htmlspecialchars($_POST['nombre'], ENT_QUOTES, 'UTF-8');
$comentario = htmlspecialchars($_POST['comentario'], ENT_QUOTES, 'UTF-8');
```

3. **Hallazgo: Ausencia de Tokens Anti-CSRF Detectada**
    - **Descripción del Hallazgo con ZAP:** OWASP ZAP identificó la "Ausencia de Tokens Anti-CSRF" en las solicitudes del formulario de comentarios. Esto indicaba que el sistema no verificaba la legitimidad de las solicitudes POST, dejándolo expuesto a ataques donde un atacante podría forzar al navegador de un usuario autenticado a realizar acciones no deseadas.

- **Corrección Aplicada:** Se implementó un token CSRF único por sesión, el cual es generado por el servidor, incluido en el formulario HTML como un campo oculto, y verificado en el lado del servidor para cada solicitud POST, invalidando aquellas que no presenten un token válido o coincidente.

```php
// Verificación de token CSRF
if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("❌ Token CSRF inválido.");
}
```